FIG. 4B excepting for three differences. In FIG. 4C the check for setting up the connection through one middle switch also efficiently implements the half of the check for setting up the connection through two middle switches. The second difference is the loop control code. In the flowchart of FIG. 4B the loop exit test is performed at the end of the inner and outer loops whereas in the flowchart of FIG. 4C the loop exit test is performed at the beginning of the inner loop and outer loops.

And the following method illustrates the psuedo code for one implementation of the scheduling method of FIG. 4C to always set up a new multicast connection request through the network of FIG. 2B, when there are at least $2 * n_1 + n_2 - 1$ middle switches in the network as discussed above.

Pseudo code of the scheduling method:

```
Step 1:   c = current connection request;
Step 2:   for i = mid_switch_1 to mid_switch_m do {
Step 3:       if (c has no available link to i) continue;
Step 4:           O_i = Set of all destination switches of c having available links from i ;
Step 5:           O_k = Set of all destination switches of c having no available links from i ;
Step 6:       if ( O_i = All the required destination switches of c) {
                  Set up c through i ;
                  Mark all the used paths to and from I as unavailable;
                  return ("SUCCESS");
              }
Step 7:       for j = mid_switch_1 to mid_switch_m do {
Step 8:           if (i = j) {
                    continue;
Step 9:           } else {
                      O_j = Set of all destination switches of c having available links from j ;
Step 10:              if ( O_k ⊆ O_j ) {
                          Set up c through i and j;
                          Mark all the used paths to and from i and j as unavailable;
                          return ("SUCCESS");
                      }
                  }
              }
```

}

Step 11: return("Never Happens");

Step 1 above labels the current connection request as "c". Step 2 starts an outer loop of a doubly nested loop and steps through all the middle switches. If the input switch of c has no available link to the middle switch i, next middle switch is selected to be i in the Step 3. Steps 4 and 5 determine the set of destination switches of c having and not having available links from middle switch i, respectively. In Step 6 if middle switch i have available links to all the destination switches of connection request c, connection request c is set up through middle switch i. And all the used links of middle switch i to output switches are marked as unavailable for future requests. Also the method returns "SUCCESS". Step 7 starts the inner loop to step through all the middle switches to search for the second middle switch, and if middle switch i is same as the middle switch j, Step 8 continues to select the next middle switch to be j. Step 9 determines the set of all destination switches having available links from middle switch j. And in Step 10, if all the links that are unavailable from middle switch i are available from middle switch j, connection request c is set up through middle switch i and middle switch j. All the used links from middle switch i and middle switch j to output switches are marked as unavailable and the method returns "SUCCESS". These steps are repeated for all the pairs of middle switches. One or two middle switches can always be found through which c can be set up, and so the control will never reach Step 11. It is easy to observe that the number of steps performed by the scheduling method is proportional to m^2, where m is the number of middle switches in the network and hence the scheduling method is of time complexity $O(m^2)$.

Table 2 shows how the steps 1-11 of the above pseudo code implement the flowchart of the method illustrated in FIG. 4C, in one particular implementation.

TABLE 2

Steps of the pseudo code of the scheduling method	Acts of Flowchart of FIG. 4C
1	301
2	301, 302, 315

3	304
4,5	305
6	306, 314
7	307, 308, 313
8	309
9	310
10	311, 312
11	303

FIG. 4D illustrates, in one embodiment, the data structures used to store and retrieve data from memory of a controller that implements the method of FIG. 4C. In this embodiment, the fan-out of at most two in the input switch of each connection is implemented by use of two data structures (such as arrays or linked lists) to indicate the destinations that can be reached from each of two middle switches. Specifically as illustrated in FIG. 4D, two arrays 530 and 550 are determined for each of the two middle switches MSi and MSj that are checked for possible use in setting up the connection, for example in act 142 of the scheduling method 140 of FIG. 1B. Arrays 530 and 550 are determined as follows. Each connection request 510 is specified by an array 520 of destination switch identifiers (and also an inlet link of an input switch identifier). Another array 560 of middle switches contains m elements one each for all the middle switches of the network. Each element of array 560 has a pointer to one of m arrays, 570-1 to 570-m, containing a bit that indicates availability status (hereinafter availability status bit) for each output switch OS1-OSr as shown in FIG. 4D. If second internal link to an output switch is available from a middle switch, the corresponding bit in the availability status array is set to 'A' (to denote available, i.e. unused link) as shown in FIG. 4D. Otherwise the corresponding bit is set to 'U' (to denote unavailable, i.e. unused link).

For each connection 510 each pair of middle switches MSi, and MSj are checked to see if all the destinations of connection 510 are reachable from the pair. Specifically this condition is checked by using the availability status arrays 570-i, 570-j of two middle switches MSi and MSj, to determine the available destinations of the connection 510 from MSi and MSj in the respective arrays 530 and 550. In one implementation, each